



WESTFÄLISCHE
WILHELMS-UNIVERSITÄT
MÜNSTER

An „almost non-intrusively“ Software Concept for Operator EIM

RBM Summer School 2011, Reisensburg



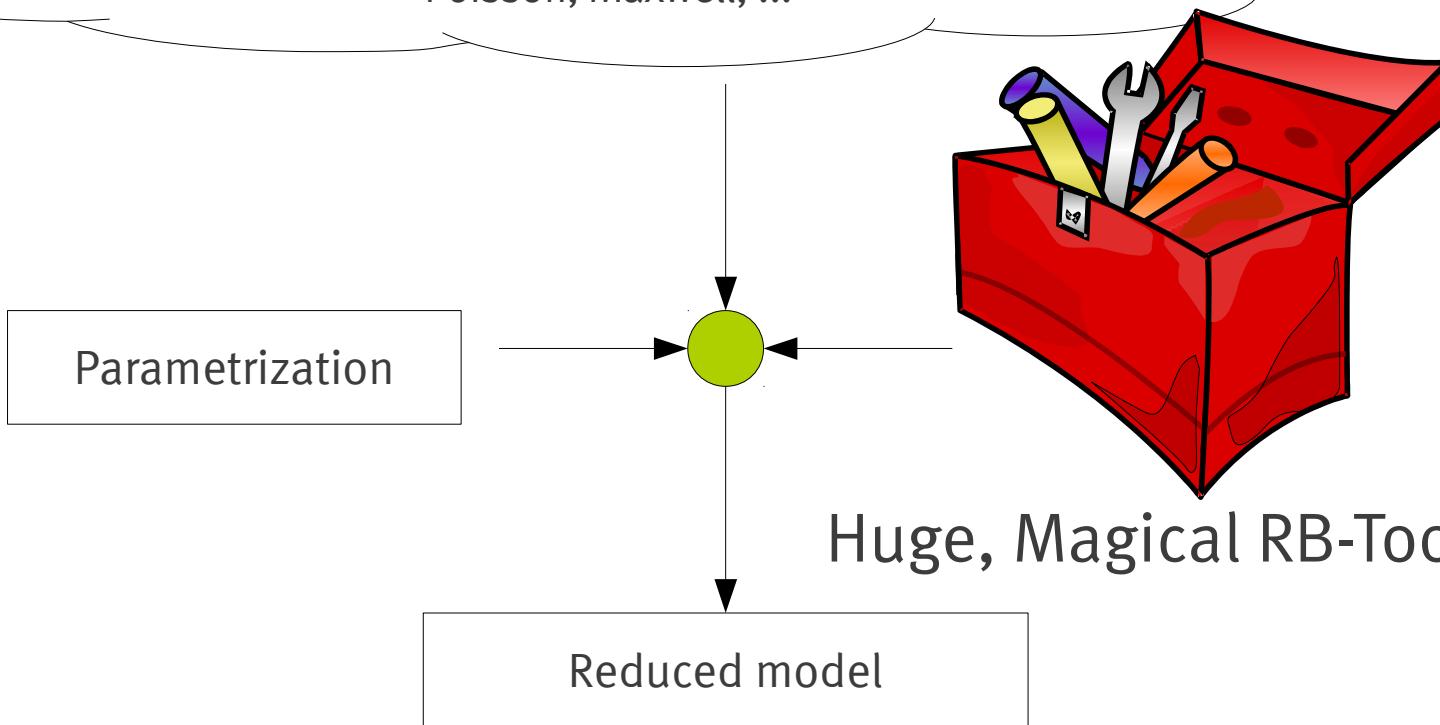
› Implementation parts of reduced basis methods

- Numerical scheme (truth/detailed solutions) + Parametrization
- Greedy Algorithm
 - POD-Greedy, EI-Greedy, PODEI-Greedy
 - p-Partitioning, t-Partitioning, hp-Partitioning
 - Training set adaptation
- Needed tools
 - Orthonormalization, POD, SCM, ...
 - Data storage, postprocessing and analysis
 - Visualization
 - (Parallelisation)
- ...



> Utopian Dream

PDE Discretizations (FEM, FV, DG):
Navier-Stokes, Groundwater-Flow, Convection-Diffusion,
Poisson, Maxwell, ...



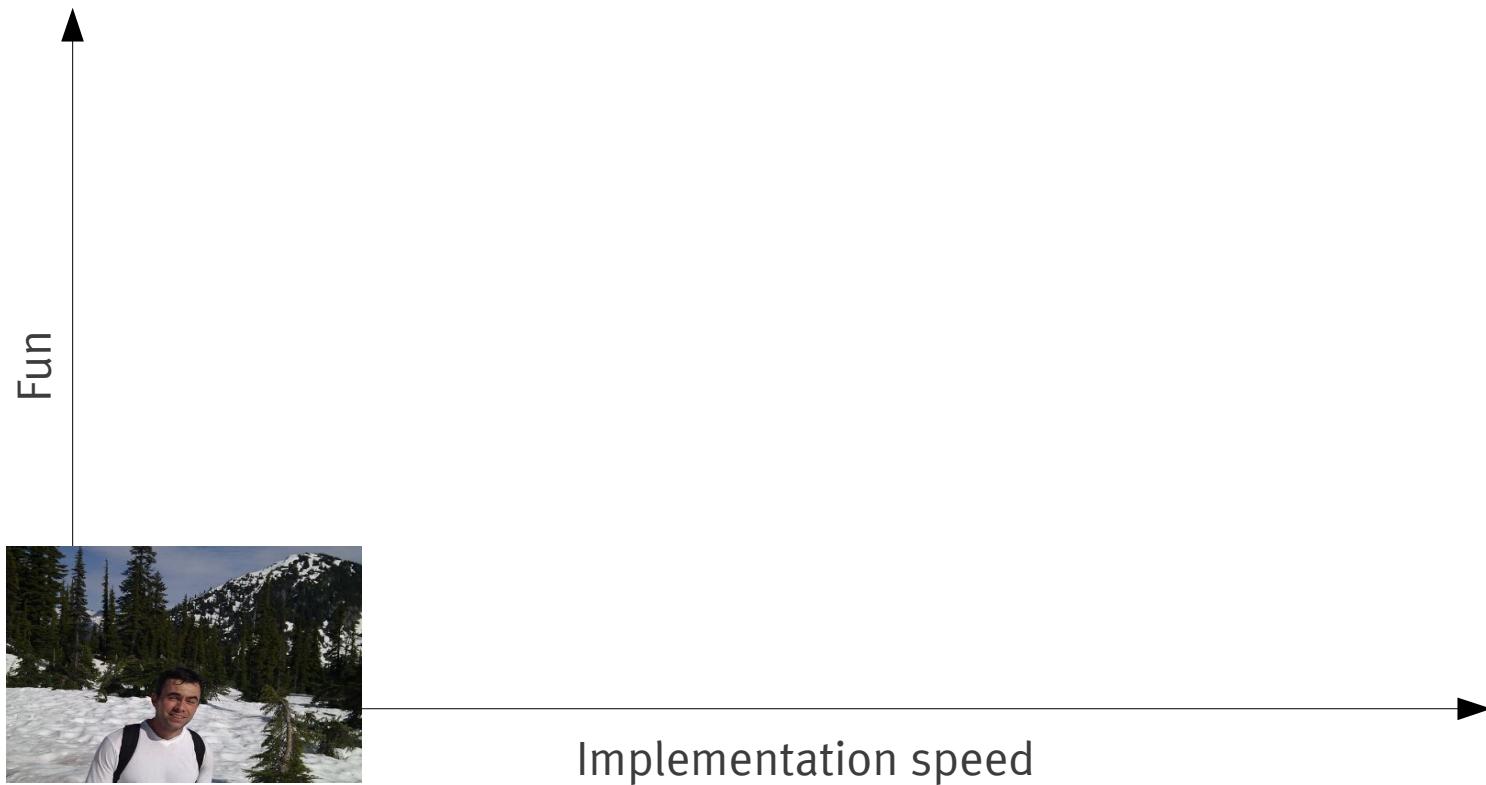
Utopian dream

- Fast application of reduced basis tools to (existing) sophisticated discretizations.
- Quick and easy implementation of new reduced basis tools or extension of existing reduced basis tools.

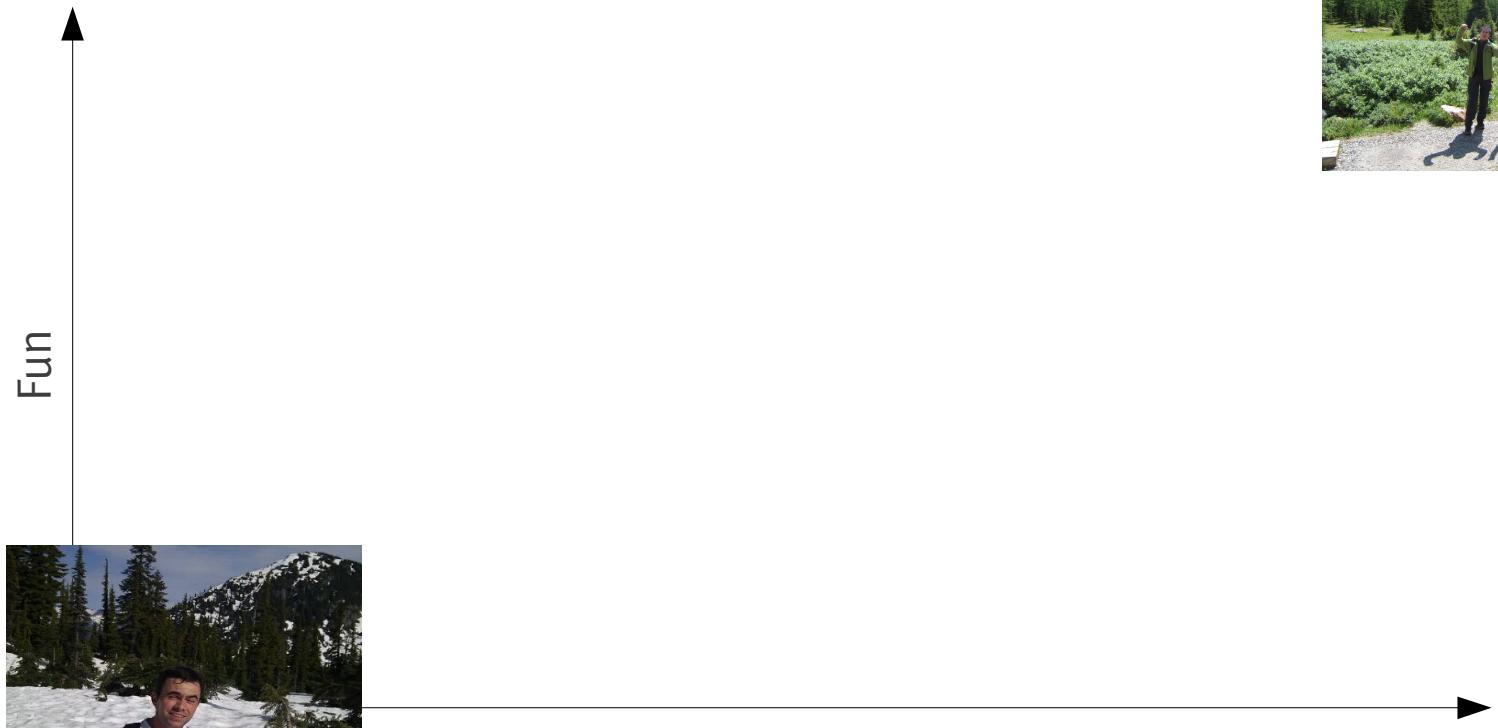




Summary

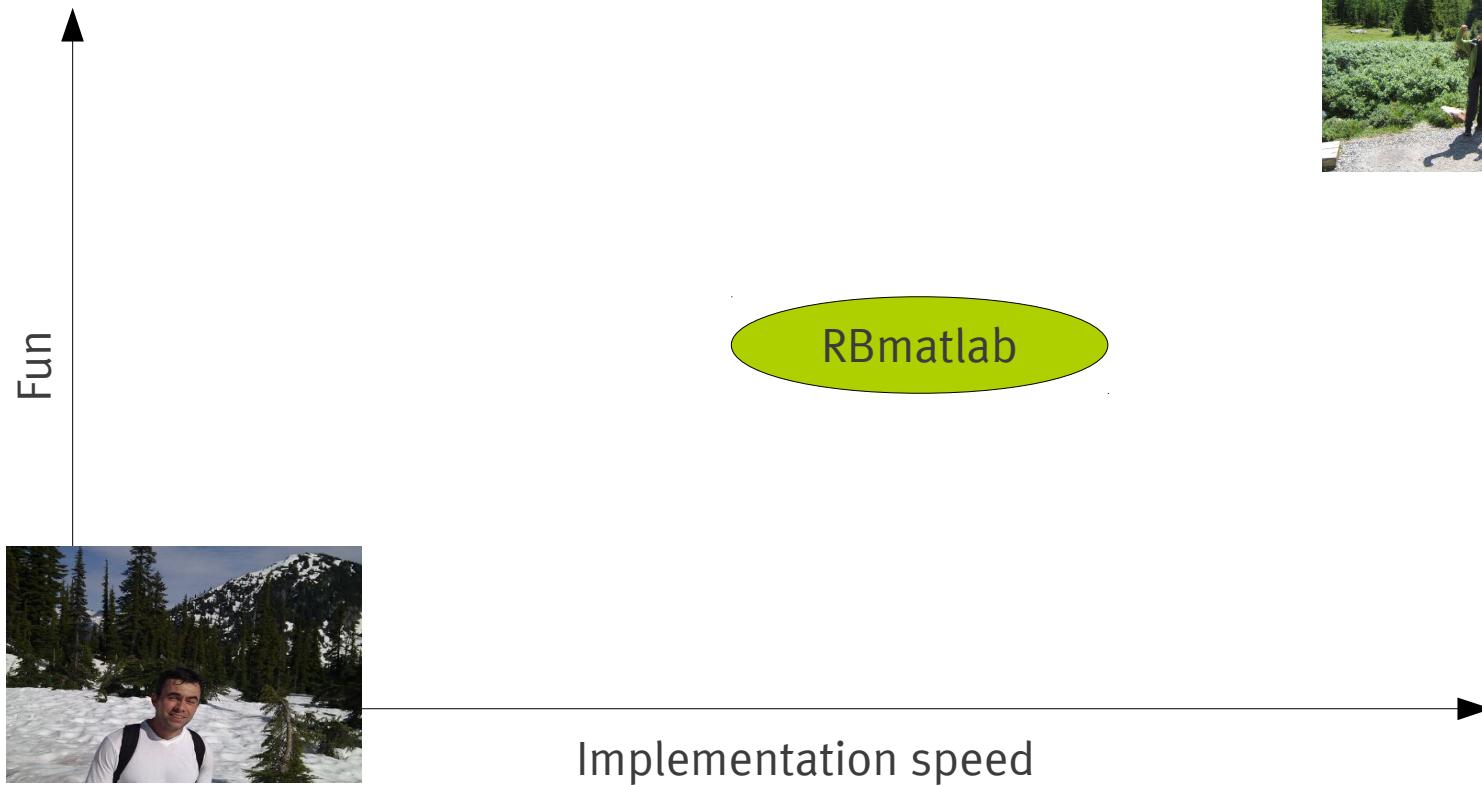


Summary





Reality: RBmatlab



Reality: RBmatlab

Fun



Implementation speed

› Outline

- RBmatlab concepts:

- General course of action for RB model reduction
- Greedy basis generation algorithms
- Operator EIM
- Combination with DUNE-RB

Download: www.morepas.org/software

The screenshot shows the MoRePaS website's "Software" page. The top navigation bar includes links for Home, Research, Software (which is the active page), News, and People. A banner image at the top right shows a heatmap or simulation results. The left sidebar features logos for RWTH Aachen University, University of Konstanz, University of Münster, University of Stuttgart, and University of Ulm. The main content area is titled "SOFTWARE" and contains text about the projects and details for the RBmatlab and Dune-RB packages.

Our projects are mainly based on two software packages, that will be made available at suitable time in the near future:

- ⦿ **RBmatlab:** A MATLAB library containing all our reduced simulation approaches for linear and nonlinear, affine or arbitrarily parameter dependent evolution problems with finite element, finite volume or local discontinuous Galerkin discretizations.
 - ⦿ Release 0.11.04 (May 04, 2011)
Documentation and installation instructions: [read online](#) or [download](#).
Tarball: [rbmatlab-release-0.11.04.tar.gz](#) (156MB)
- ⦿ **Dune-RB:** A module for the Dune library ([www.dune-project.org](#), [http://dune.mathematik.uni-freiburg.de](#)), which realizes C++ template classes for use in snapshot generation and RB offline phases for various discretizations. Apart from single-core algorithms, the package also aims at using parallelization techniques for efficient snapshot generation.



RBmatlab main parts

Data structures:

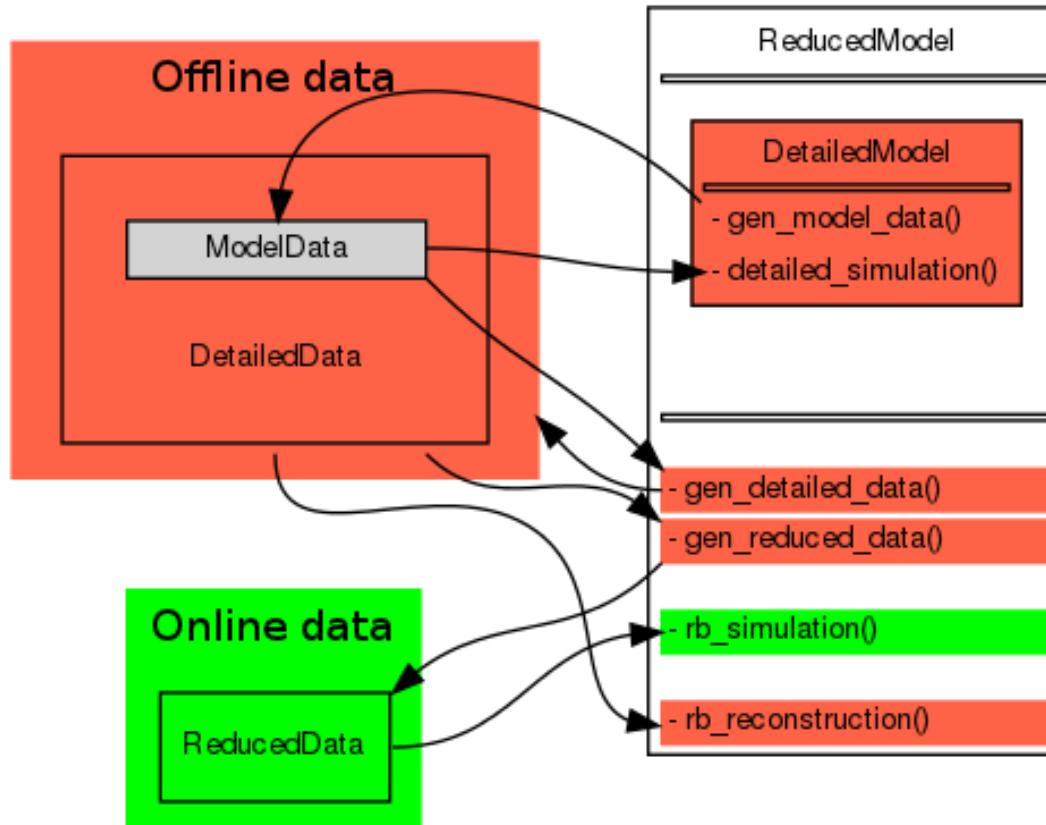
model_data	structure with high-dimensional data for detailed computations
detailed_data	class object with high-dimensional data created during offline phase
reduced_data	class object with low-dimensional data created during offline phase

Models:

DetailedModel class with methods for detailed simulations (wrapper for discretization)

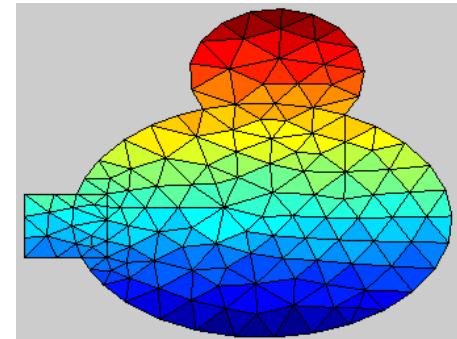
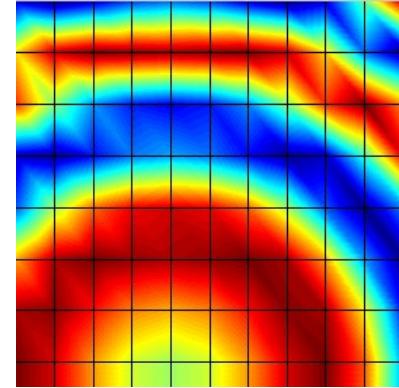
ReducedModel Class with methods for basis generation and reduced simulations

RBmatlab course of action



RBmatlab discretization tools

- Grid implementations
 - 2D: triangular grid, rectangular
 - nD: cubical grid
- Discrete function spaces
 - FV, FE, LDG
- (Operators and data functions)



Example: FV scheme for Burgers-Equation

Burgers Equation

$$\partial_t u - \nabla \mathbf{v} u^{\mu_1} = 0 \quad (1)$$

with (implicit) finite volume discretization with Engquist Osher flux.

- ▶ Parameter vector $\mu := (\mu_1) \in [1, 2]$.
- ▶ $\Omega = [0, 2] \times [0, 1]$ with purely cyclical boundary conditions
- ▶ end time $T = 0.3$
- ▶ smooth initial data: $u_0(x) = \frac{1}{2}(1 + \sin(2\pi x_1) \sin(2\pi x_2))$
- ▶ rectangular 120×60 grid with $K = 100$ time steps.

Example: FV scheme for Burgers-Equation

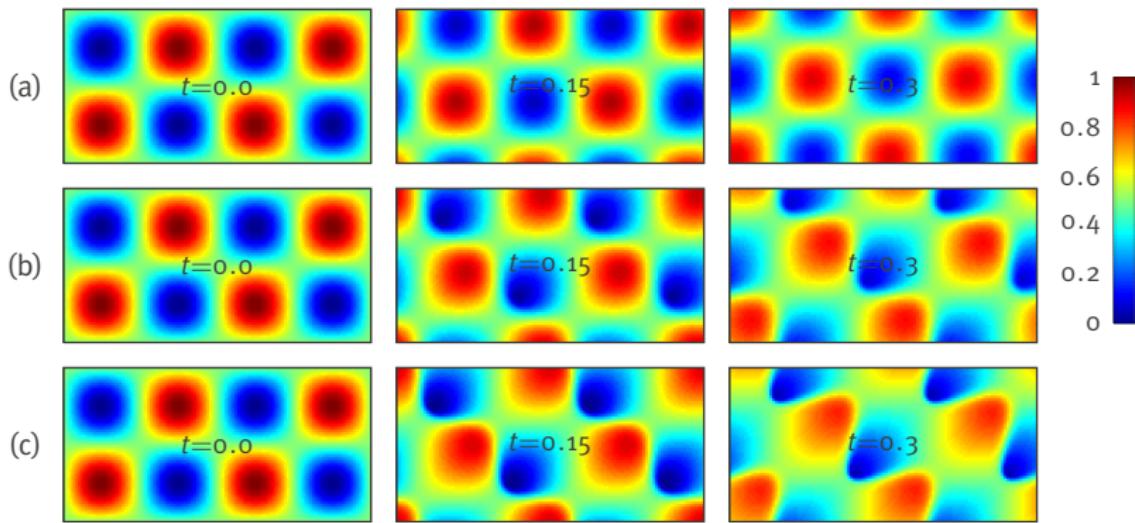


Figure: Illustration of transport for smooth data. Snapshots at different time instants for a) $\mu = 1$ b) $\mu = 1.5$ and c) $\mu = 2$.

Discretization (implicit/explicit with Newton scheme)

For $\mu \in \mathcal{P}$ find $\{u_h\}_{k=0}^K \subset \mathcal{W}_h \subset \mathcal{W}$, s.t.

$$u_h^0 := \mathcal{P}_h [u_0(\mu)], \quad u_h^{k+1} := u_h^{k+1, \nu_{\max}(k)}$$

with Newton iteration

$$\begin{aligned} u_h^{k+1,0} &:= u_h^k, & u_h^{k+1,\nu+1} &:= u_h^{k+1,\nu} + \delta_h^{k+1,\nu+1}, \\ \left(\text{Id} + \Delta t \mathbf{D} \mathcal{L}_{h,I} \Big|_{u_h^{k+1,\nu}} \right) [\delta_h^{k+1,\nu+1}] &= u_h^k - u_h^{k+1,\nu} - \Delta t \left(\mathcal{L}_{h,I} [u_h^{k+1,\nu}] + \mathcal{L}_{h,E} [u_h^k] \right). \end{aligned}$$

Reduced basis scheme

Reduced simulation

For $\mu \in \mathcal{P}$ find $\{u_{\text{red}}^k\}_{k=0}^K \subset \mathcal{W}_{\text{red}}$, s.t.

$$u_{\text{red}}^{k+1} := u_{\text{red}}^{k+1, \nu_{\max}(k)}, \quad u_{\text{red}}^0 := \mathcal{P}_{\text{red}}[u_{h,0}(\mu)]$$

with Newton iteration

$$u_{\text{red}}^{k+1,0} := u_{\text{red}}^k, \quad u_{\text{red}}^{k+1,\nu+1} := u_{\text{red}}^{k+1,\nu} + \delta_{\text{red}}^{k+1,\nu+1},$$
$$\left(\text{Id} + \Delta t \mathbf{D}\mathcal{L}_{\text{red},I} |_{u_{\text{red}}^{k+1,\nu}} \right) [\delta_{\text{red}}^{k+1,\nu+1}] = u_{\text{red}}^k - u_{\text{red}}^{k+1,\nu} - \Delta t \left(\mathcal{L}_{\text{red},I} [u_{\text{red}}^{k+1,\nu}] + \mathcal{L}_{\text{red},E} [u_{\text{red}}^k] \right).$$

Prerequisites

- ▶ $\mathcal{P}_{\text{red}} : \mathcal{W}_h \rightarrow \mathcal{W}_{\text{red}}$: Galerkin projection onto $\mathcal{W}_{\text{red}} \subset \mathcal{W}_h$

- ▶ $\mathcal{L}_{\text{red},I} := \mathcal{P}_{\text{red}} \circ \mathcal{I}_M \circ \mathcal{L}_{h,I}$: implicit operator evaluation
- ▶ $\mathcal{L}_{\text{red},E} := \mathcal{P}_{\text{red}} \circ \mathcal{I}_M \circ \mathcal{L}_{h,E}$: explicit operator evaluation

Prerequisites

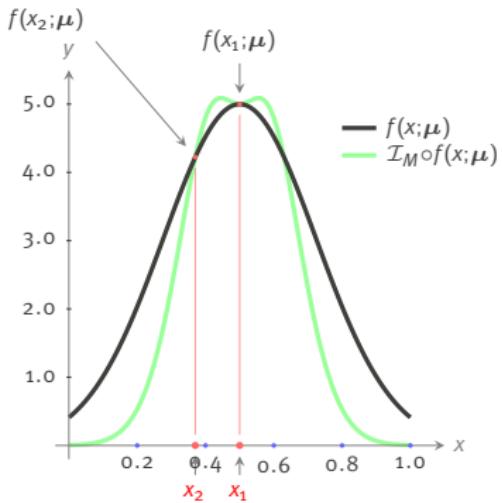
- ▶ $\mathcal{P}_{\text{red}} : \mathcal{W}_h \rightarrow \mathcal{W}_{\text{red}}$: Galerkin projection onto $\mathcal{W}_{\text{red}} \subset \mathcal{W}_h$

Empirical interpolation of discrete operators

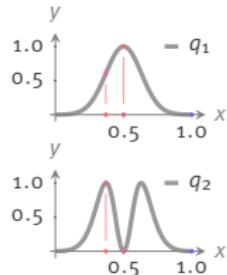
- ▶ $\mathcal{L}_{\text{red},I} := \mathcal{P}_{\text{red}} \circ \mathcal{I}_M \circ \mathcal{L}_{h,I}$: implicit operator evaluation
- ▶ $\mathcal{L}_{\text{red},E} := \mathcal{P}_{\text{red}} \circ \mathcal{I}_M \circ \mathcal{L}_{h,E}$: explicit operator evaluation

Empirical interpolation[Barrault et al, 2004]

Empirical interpolation for parametrized functions $f(\mu) : \mathbb{R} \rightarrow \mathbb{R}$.



Base functions:

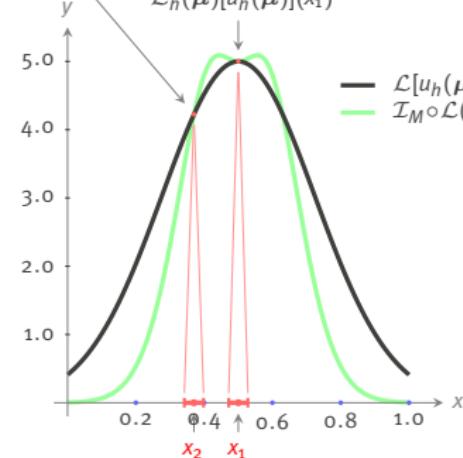


- ▶ “magic points” $\{x_m\}_{m=1}^M$
- ▶ basis functions $\{q_m\}_{m=1}^M$

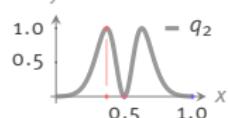
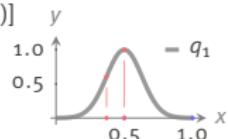
Empirical interpolation [Barrault et al, 2004]

Empirical interpolation for parametrized discrete operators $\mathcal{L}_h \in \mathcal{W}_h$.

$$\mathcal{L}_h(\mu)[u_h(\mu)](x_2)$$



Base functions:

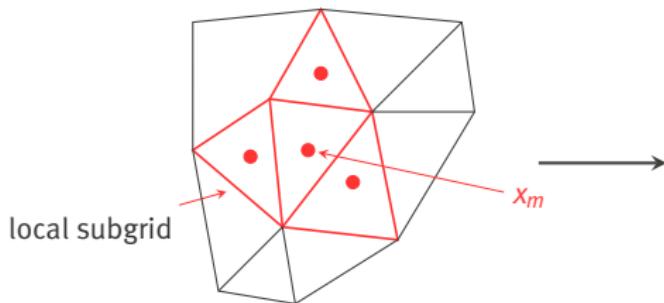


- ▶ “magic points” $\{x_m\}_{m=1}^M$
- ▶ basis functions $\{q_m\}_{m=1}^M$

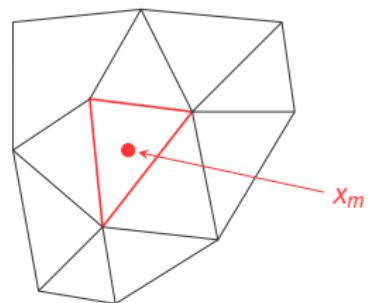
Discrete operators need to have “*H-independent Dof dependence*”.

Empirical interpolation: Subgrids

Restrict $u_h(\mu)$ to subgrid



Evaluate $\mathcal{L}_h(\mu)[u_h(\mu)](x_m)$



Efficient evaluations

The operator evaluations in interpolation points $\mathcal{L}_h(\mu)[\cdot](x_m)$ can be computed efficiently during **online** phase, if

- ▶ the operator has a localized structure (**small stencil**) and
- ▶ the local geometry information is precomputed during **offline** phase.

Empirical interpolation: Details

General operator approximation

Interpolate operator evaluations at “magic points”:

parameter de-
pendent

parameter inde-
pendent

$$\mathcal{L}_h(\mu) \left[u_h^k(\mu) \right] (x_m) = (\mathcal{I}_M \circ \mathcal{L}_h(\mu)) \left[u_h^k(\mu) \right] (x_m) = \sum_{m=1}^M \sigma_m(\mu) q_m(x_m).$$

Empirical interpolation

- ▶ Basis functions q_m are directly computed from operator evaluations
- ▶ for selected parameters μ_m and time steps k_m .
- ▶ They span a collateral reduced basis space $\mathcal{W}_M \subset \mathcal{W}_h$.

Empirical interpolation: Fréchet derivative

Define $l_m(\mu) : \mathcal{W}_h \rightarrow \mathbb{R}$, $u_h \mapsto \mathcal{L}_h(\mu)[u_h](x_m)$

Observation

$$(\mathbf{D} (\mathcal{I}_M [\mathcal{L}_h(\mu)])|_{u_h} [v_h])(x_m) = (\mathbf{D} l_m(\mu)|_{u_h} [v_h])$$

Empirical interpolation: Fréchet derivative

Define $l_m(\mu) : \mathcal{W}_h \rightarrow \mathbb{R}, u_h \mapsto \mathcal{L}_h(\mu)[u_h](x_m)$

$$(\mathbf{D}(\mathcal{I}_M[\mathcal{L}_h(\mu)])|_{u_h}[v_h])(x_m) = (\mathbf{D}l_m(\mu)|_{u_h}[v_h])$$

Coefficient functionals

\mathcal{W}_h is finite dimensional

$$\mathbf{D}l_m(\mu)|_{u_h}[v_h] = \sum_{m=1}^M \sum_{i=1}^H \frac{\partial}{\partial \psi_i} l_m(\mu)[u_h](v_{h,i})$$

Empirical interpolation: Fréchet derivative

Define $l_m(\mu) : \mathcal{W}_h \rightarrow \mathbb{R}, u_h \mapsto \mathcal{L}_h(\mu)[u_h](x_m)$

$$(\mathbf{D}(\mathcal{I}_M[\mathcal{L}_h(\mu)])|_{u_h}[v_h])(x_m) = (\mathbf{D}l_m(\mu)|_{u_h}[v_h])$$

Coefficient functionals

\mathcal{W}_h is finite dimensional

$$\mathbf{D}l_m(\mu)|_{u_h}[v_h] = \sum_{m=1}^M \sum_{i=1}^H \frac{\partial}{\partial \psi_i} l_m(\mu)[u_h](v_{h,i})$$

$$= \sum_{m=1}^M \sum_{\substack{i \in I_{x_m}}} \frac{\partial}{\partial \psi_i} l_m(\mu)[u_h](v_{h,i}).$$

\mathcal{L}_h has local stencil

Empirical interpolation: Fréchet derivative

Define $l_m(\mu) : \mathcal{W}_h \rightarrow \mathbb{R}$, $u_h \mapsto \mathcal{L}_h(\mu)[u_h](x_m)$

$$(\mathbf{D}(\mathcal{I}_M[\mathcal{L}_h(\mu)])|_{u_h}[v_h])(x_m) = (\mathbf{D}l_m(\mu)|_{u_h}[v_h])$$

Coefficient functionals

\mathcal{W}_h is finite dimensional

$$\mathbf{D}l_m(\mu)|_{u_h}[v_h] = \sum_{m=1}^M \sum_{i=1}^H \frac{\partial}{\partial \psi_i} l_m(\mu)[u_h](v_{h,i})$$

$$= \sum_{m=1}^M \sum_{\substack{i \in I_{x_m}}} \frac{\partial}{\partial \psi_i} l_m(\mu)[u_h](v_{h,i}).$$

\mathcal{L}_h has local stencil

Note: $\text{card}(I_{x_m}) < C$ for all $m = 1, \dots, M$. \Rightarrow Complexity still $\mathcal{O}(M)$.

Summary: Reduced basis scheme

- ▶ Generate reduced basis space with **POD-Greedy** algorithm:
 $\mathcal{W}_{\text{red}} := \text{span} \{ \varphi_i \}_{i=1}^N \subset \mathcal{W}_h$
- ▶ Reduced model order by Galerkin projection: $\mathcal{P}_{\text{red}} : \mathcal{W}_h \rightarrow \mathcal{W}_{\text{red}}$
- ▶ Offline-/online decomposition of operators:

$$(\mathcal{P}_{\text{red}}[u_{h,0}(\mu)])_n = \underbrace{\sum_{q=1}^{Q_{uo}} \sigma_{uo}^q(\mu)}_{\text{online}} \underbrace{\int_{\Omega} u_0^q \varphi_n}_{\text{offline}} \quad \text{assuming: } u_0(\mu) = \sum_{q=1}^{Q_{uo}} \sigma_{uo}^q(\mu) u_0^q$$

$$\left(\mathcal{L}_{\text{red},I}(\mu) \begin{bmatrix} u_{\text{red}}^k(\mu) \end{bmatrix} \right)_{nm} = \underbrace{\sum_{m=1}^M "l_m^I(\mu) [u_{\text{red}}]''}_{\text{online}} \underbrace{\int_{\Omega} q_m \varphi_n}_{\text{offline}}$$

$$\left(\mathbf{D}\mathcal{L}_{\text{red},I}(\mu)|_{u_{\text{red}}} [\delta_{\text{red}}] \right)_{nm} = \underbrace{\sum_{m=1}^M "\frac{\partial}{\partial \psi_i} l_m^I(\mu) [u_{\text{red}}]''}_{\text{online}} \underbrace{\int_{\Omega} q_m \varphi_n}_{\text{offline}}$$

X-greedy algorithm

X-GREEDY(M_{train} , ε_{tol} , Υ_{\max})

– Initialize reduced basis of dimension Υ_0 :

$$\begin{aligned}\mathcal{D}_{\Upsilon_0} &\leftarrow \text{X-INITBASIS0} \\ \Upsilon &\leftarrow \Upsilon_0\end{aligned}$$

repeat

– Find worst approximated parameter:

$$(\mu_{\max}, t_{\max}) \leftarrow \arg \max_{\mu \in M_{\text{train}}} \text{X-ERRORESTIMATE}(\mathcal{D}_{\Upsilon}, \mu, t^k)$$

– Extend reduced basis by snapshot:

$$\begin{aligned}\mathcal{D}_{\Upsilon+1} &\leftarrow \text{X-EXTENDBASIS}(\mathcal{D}_{\Upsilon}, \mu_{\max}, t_{\max}) \\ \Upsilon &\leftarrow \Upsilon + 1\end{aligned}$$

until $\max_{\mu \in M_{\text{train}}} \text{X-ERRORESTIMATE}(\mathcal{D}_{\Upsilon}) \leq \varepsilon_{\text{tol}}$ or $\Upsilon > \Upsilon_{\max}$

return reduced data: \mathcal{D}_{Υ}

- **Extension:** Adaptive extension of Parameter sampling M_{train} .

EI-greedy methods (1/2)

EI-INITBASIS()

```
return empty initial basis:  $\mathcal{D}_0 \leftarrow \{\}$ 
```

EI-ERRORESTIMATE($(\mathbf{Q}_M, \Sigma_M), \mu, t^k$)

- Compute exact operator evaluation
 $v_h \leftarrow \mathcal{L}_h[u_h^k(\mu)]$
 - Compute empirical interpolated operator evaluation
 $v_M \leftarrow \mathcal{I}_M \circ \mathcal{L}_h[u_h^k(\mu)]$
- return approximation error: $\|v_h - v_M\|.$

EI-greedy methods (1/2)

EI-INITBASIS()

```
return empty initial basis:  $\mathcal{D}_0 \leftarrow \{\}$ 
```

EI-ERRORESTIMATE($(\mathbf{Q}_M, \Sigma_M), \mu, t^k$)

- Compute exact operator evaluation

$$v_h \leftarrow \mathcal{L}_h[u_h^k(\mu)]$$

- Compute empirical interpolated operator evaluation

$$v_M \leftarrow \mathcal{I}_M \circ \mathcal{L}_h[u_h^k(\mu)]$$

return approximation error: $\|v_h - v_M\|.$

Detailed simulation for all trainings parameters needed. **Expensive!**

EI-greedy methods (2/2)

EI-EXTENDBASIS((\mathbf{Q}_M, Σ_M) , μ, t^k)

- Compute exact operator evaluation.
 $v_h \leftarrow \mathcal{L}_h[u_h^k(\mu)]$
- Compute empirical interpolated operator evaluation.
 $v_M \leftarrow \mathcal{I}_M \circ \mathcal{L}_h[u_h^k(\mu)]$
- Compute the residual.
 $r_M \leftarrow v_h - v_M.$
- Find “magic point” maximizing the residual.
 $x_{M+1} \leftarrow \arg \sup_{x \in \Sigma_h} |r_M(x)|$
- Normalize to obtain a new basis function.
 $q_{M+1} \leftarrow r_M \cdot \frac{1}{r_M(x_{M+1})}$

return extended basis data: $\mathcal{D}_{M+1} \leftarrow \left(\{q_m\}_{m=1}^{M+1}, \{x_m\}_{m=1}^{M+1} \right)$

POD-greedy methods

POD-INITBASIS()

return initial reduced basis functions: $\{\varphi_n\}_{n=1}^{N_0}$

POD-ERRORESTIMATE($\{\varphi_n\}_{n=1}^N, \mu, t^k$)

return error estimate: $\eta_{N,M}^k(\mu) \geq \|u_{\text{red}}^k(\mu) - u_h^k(\mu)\|$

POD-EXTENDBASIS($\{\varphi_n\}_{n=1}^N, \mu_{\max}, \cdot$)

- Compute trajectory $\{u_h^k(\mu_{\max})\}_{k=0}^K$.
- Compute new basis function with POD and Galerkin projection \mathcal{P}_{red} projecting onto span $\{\varphi_n\}_{n=1}^N$:

$$\varphi_{N+1} \leftarrow \text{POD} \left(\{u_h^k(\mu_{\max}) - \mathcal{P}_{\text{red}}[u_h^k(\mu_{\max})]\}_{k=0}^K \right)$$

return extended reduced basis: $\{\varphi_n\}_{n=1}^{N+1}$

POD-greedy methods

POD-INITBASIS()

return initial reduced basis functions: $\{\varphi_n\}_{n=1}^{N_0}$

POD-ERRORESTIMATE($\{\varphi_n\}_{n=1}^N, \mu, t^k$)

return *error estimate*: $\eta_{N,M}^k(\mu) \geq \|u_{\text{red}}^k(\mu) - u_h^k(\mu)\|$

POD-EXTENDBASIS($\{\varphi_n\}_{n=1}^N, \mu_{\max}, \cdot$)

- Compute trajectory $\{u_h^k(\mu_{\max})\}_{k=0}^K$.
- Compute new basis function with POD and Galerkin projection \mathcal{P}_{red} projecting onto span $\{\varphi_n\}_{n=1}^N$:

$$\varphi_{N+1} \leftarrow \text{POD} \left(\{u_h^k(\mu_{\max}) - \mathcal{P}_{\text{red}}[u_h^k(\mu_{\max})]\}_{k=0}^K \right)$$

return extended reduced basis: $\{\varphi_n\}_{n=1}^{N+1}$

Remarks: EI-greedy + POD-greedy

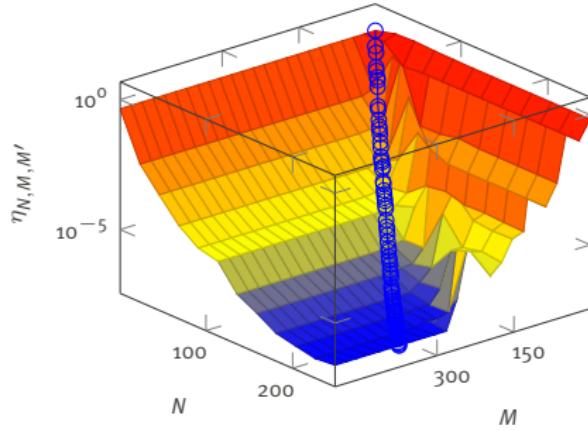
- ▶ “EI-greedy” has to be computed before “POD-greedy” with detailed simulations for all trainings parameters.
- ▶ A priori it is unknown, how to choose the error tolerance ε_{tol} for the “EI-greedy”.
- ▶ Number of EI base functions might be too large.

Remarks: EI-greedy + POD-greedy

- ▶ “EI-greedy” has to be computed before “POD-greedy” with detailed simulations for all trainings parameters.
- ▶ A priori it is unknown, how to choose the error tolerance ε_{tol} for the “EI-greedy”.
- ▶ Number of EI base functions might be too large.

Alternative: “PODEI-greedy”

Motivation: PODEI-greedy



Example: Burgers Equation

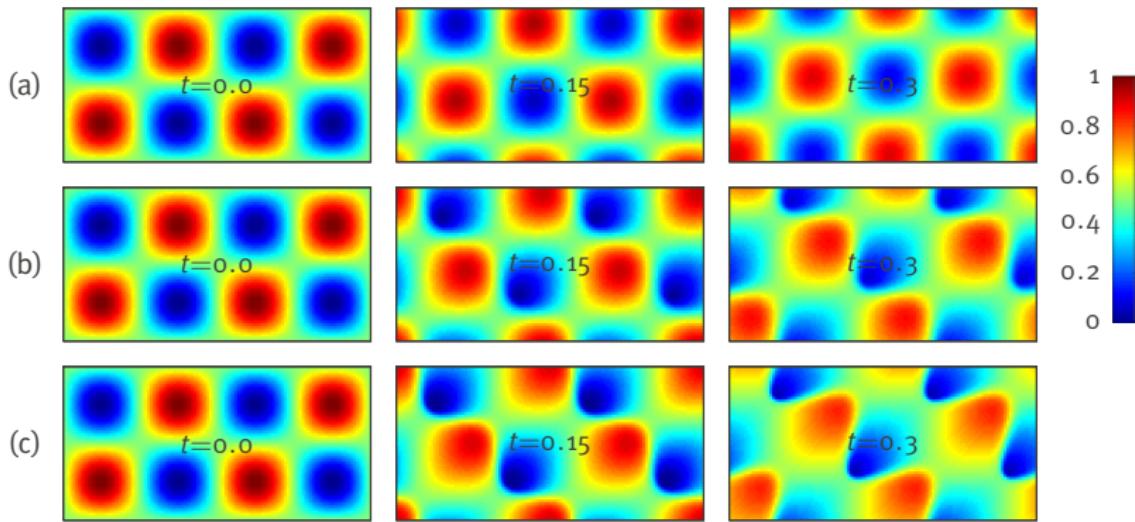
Burgers Equation

$$\partial_t u - \nabla \mathbf{v} u^{\mu_1} = 0 \quad (2)$$

with (implicit) finite volume discretization with Engquist Osher flux.

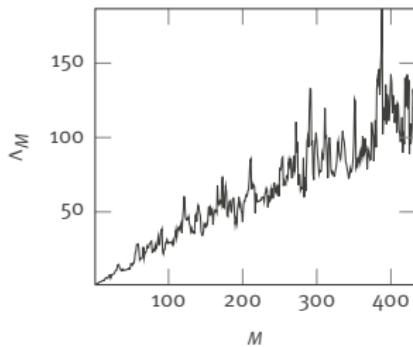
- ▶ Parameter vector $\mu := (\mu_1) \in [0, 2]$.
- ▶ $\Omega = [0, 2] \times [0, 1]$ with purely cyclical boundary conditions
- ▶ end time $T = 0.3$
- ▶ smooth initial data: $u_0(x) = \frac{1}{2}(1 + \sin(2\pi x_1) \sin(2\pi x_2))$
- ▶ rectangular 120×60 grid with $K = 100$ time steps.

Example: Solution snapshots



Example: Empirical interpolation of $\mathcal{L}_{h,I}$

(a) Lebesgue constant



(b) El-greedy

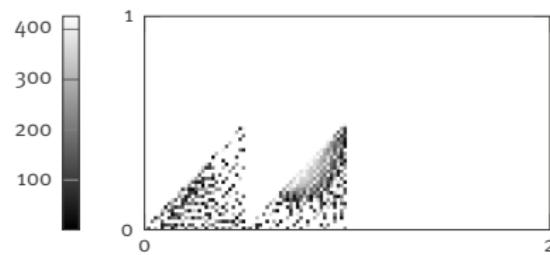


Illustration of interpolation DOF selection for Burgers problem. DOFs corresponding to darker points are selected first.

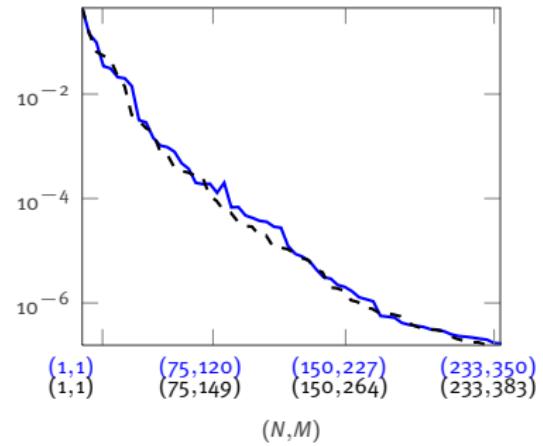
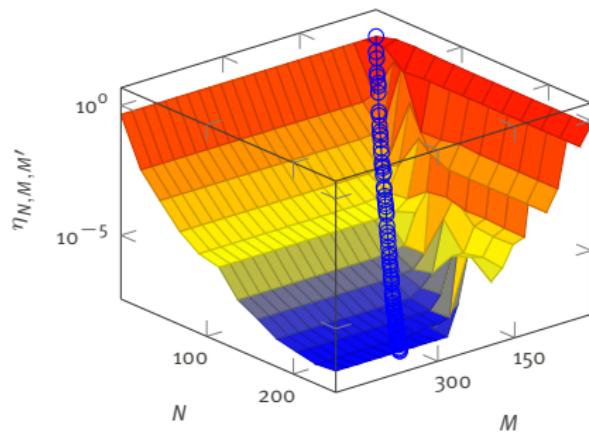
Example: Table

- ▶ $\dim(\mathcal{W}_h)$ 9600
- ▶ $\nu_{\max} \approx 1 - 20$
- ▶ $\#M_{\text{train}}$ 28

N	M	$\varnothing\text{-runtime}[s]$	max. error	$\varnothing\text{-offline time}[h]$
7,200	0	90.01	0.00	0
42	83	4.42	$1.15 \cdot 10^{-3}$	0.96
83	166	6.23	$6.03 \cdot 10^{-5}$	1.34
125	250	8.99	$7.43 \cdot 10^{-6}$	1.74
166	333	11.6	$8.33 \cdot 10^{-7}$	2.23
208	416	15.64	$2.47 \cdot 10^{-7}$	2.78
249	499	19.56	$2.38 \cdot 10^{-7}$	3.4

N	M	$\varnothing\text{-runtime}[s]$	max. error	$\varnothing\text{-offline time}[h]$
0	-1	90.01	0.00	0
42	72	4.44	$1.73 \cdot 10^{-3}$	0.54
83	144	6.04	$5.74 \cdot 10^{-5}$	1.09
125	216	8.37	$7.30 \cdot 10^{-6}$	1.55
167	288	11.92	$7.63 \cdot 10^{-7}$	2.08
208	360	15.08	$2.31 \cdot 10^{-7}$	2.69
233	402	16.48	$1.55 \cdot 10^{-7}$	3.27

Example: Error landscape



Thank you for your attention!

› Low dimensional Computations (

RBMATLAB

)

- Control of Offline-Algorithms
 - POD-Greedy Algorithm
 - Gathering and post-processing of reduced matrices
- Low-Dimensional Computations
 - Reduced Simulations
 - A posteriori error estimation
- Low dimensional data visualization

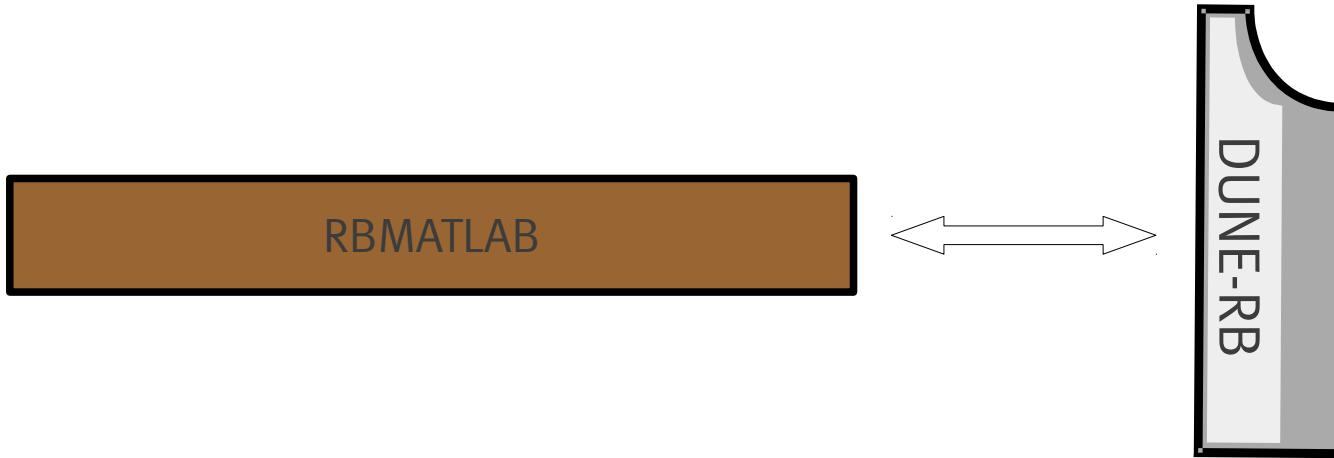


High dimensional computations (DUNE-RB)

- Storage / Manipulation of reduced spaces
- Efficient high dimensional linear algebra algorithms
 - POD, Orthonormalization, Gram-Matrix computations
- Parametrization
- (Implemented in C++, based on
 - DUNE core modules (<http://dune-project.org>) and
 - DUNE-FEM (dune.mathematik.uni-freiburg.de)

The Glue

- Communication between DUNE-RB and RBMATLAB can be realized by
 1. compiling DUNE-RB as (mex-) library for matlab or:
 2. TCP/IP Communication between two stand-alone applications



Proof of Concept (linear evolution problem)

RBMATLAB > matlab

```
< M A T L A B (R) >
Copyright 1984-2008 The MathWorks, Inc.
Version 7.6.0.324 (R2008a)
February 10, 2008
```

To get started, type one of these: helpwin, helpdesk, or demo

For product information, visit www.mathworks.com.

```
starting up rbmatlab in directory:
/home/martin/projects/rbm-results/rbmatlab
Using the following directory for large temporary data:
/tmp
Using the following directory for data files storing results:
/home/martin/projects/rbm-results/results
Using the following directory as RBMATLABHOME:
/home/martin/projects/rbm-results/rbmatlab
skipped clearing filecache for function-calls!
>> □
```

DUNE-RB > ./dunerbServer

```
YaspGridParameterBlock: Parameter 'overlap' not specified, defaulting to '0'.
```

```
server: waiting for connections...
```

Proof of Concept (linear evolution problem)

```
< M A T L A B (R) >
Copyright 1984-2008 The MathWorks, Inc.
Version 7.6.0.324 (R2008a)
February 10, 2008
```

```
To get started, type one of these: helpwin, helpdesk, or demo
For product information, visit www.mathworks.com.
```

```
starting up rbmatlab in directory:
/home/martin/projects/rbm-results/rbmatlab
Using the following directory for large temporary data:
/tmp
Using the following directory for data files storing results:
/home/martin/projects/rbm-results/results
Using the following directory as RBMATLABHOME:
/home/martin/projects/rbm-results/rbmatlab
skipped clearing filecache for function-calls!
```

```
>>
>> [a,b,c] = ...
mexclient('echo', [1, 2], ....
           struct('field', [1,2]), ...
           { [1,2], [3,4] });

```

```
client: connect: Connection refused
Warning: connection to ::1 failed
```

```
client connected to 127.0.0.1
copying argument no. 0
copying argument no. 1
copying argument no. 2
>> 
```

```
DUNE-RB > ./dunerbServer
YaspGridParameterBlock: Parameter 'overlap' not specified, defaulting to '0'.
server: waiting for connections...
server: got connection from 127.0.0.1

Received call for processing 'echo'
with 4 arguments and 3 return values.
=====
copying argument no. 0
copying argument no. 1
copying argument no. 2
```

Proof of Concept (linear evolution problem)

```
/home/martin/projects/rbm-results/rbmatlab
skipped clearing filecache for function-calls!
>>
>> [a,b,c] = ...
mexclient('echo', [1, 2], ....
           struct('field', [1,2]), ...
           { [1,2], [3,4] });
client: connect: Connection refused
Warning: connection to ::1 failed
```

```
client connected to 127.0.0.1
copying argument no. 0
copying argument no. 1
copying argument no. 2
```

```
>> a
a =
    1     2
>> b
b =
    field: [1 2]
>> c
c =
    [1x2 double]    [1x2 double]
```

```
>> █
```

```
DUNE-RB > ./dunerbServer
YaspGridParameterBlock: Parameter 'overlap' not specified, defa
ulting to '0'.
server: waiting for connections...
server: got connection from 127.0.0.1
Received call for processing 'echo'
with 4 arguments and 3 return values.
=====
copying argument no. 0
copying argument no. 1
copying argument no. 2
█
```

Proof of Concept (linear evolution problem)

```
< M A T L A B (R) >
Copyright 1984-2008 The MathWorks, Inc.
Version 7.6.0.324 (R2008a)
February 10, 2008
```

To get started, type one of these: helpwin, helpdesk, or demo
For product information, visit www.mathworks.com.

```
starting up rbmatlab in directory:
/home/martin/projects/rbm-results/rbmatlab
Using the following directory for large temporary data:
/tmp
Using the following directory for data files storing results:
/home/martin/projects/rbm-results/results
Using the following directory as RBMATLABHOME:
/home/martin/projects/rbm-results/rbmatlab
skipped clearing filecache for function-calls!
```

```
>>
>>
>> % load model parameters
>>
>> model = convdiff_dune_model;
Warning: Name is nonexistent or not a directory: mexclient.
> In path at 110
  In addpath at 87
  In convdiff_dune_model at 95
client: connect: Connection refused
Warning: connection to ::1 failed

client connected to 127.0.0.1
>> []
```

```
DUNE-RB > ./dunerbServer
YaspGridParameterBlock: Parameter 'overlap' not specified, defaulting to '0'.
server: waiting for connections...
server: got connection from 127.0.0.1
```

```
Received call for processing 'init_model'
with 1 arguments and 1 return values.
=====
```

```
read discfunclist_xdr from headerfile, size = 20
Using the explicit ode solver! In order to use a different discretization, change the 'DISCRETIZATION' make variable
```

```
Received call for processing 'get_mu'
with 1 arguments and 1 return values.
=====
```

```
Received call for processing 'rb_symbolic_coefficients'
with 1 arguments and 1 return values.
=====
```

Proof of Concept (linear evolution problem)

```
> In path at 110
  In addpath at 87
  In convdiff_dune_model at 95
client: connect: Connection refused
Warning: connection to ::1 failed
```

```
client connected to 127.0.0.1
```

```
>> model.rb_problem_type
```

```
ans =
```

```
lin_evol
```

```
>> model.RB_generation_mode
```

```
ans =
```

```
greedy_uniform_fixed
```

```
>> model.RB_stop_Nmax
```

```
ans =
```

```
20
```

```
>> model.T % this is read from DUNE-RB <<<<<
```

```
ans =
```

```
1
```

```
>> █
```

```
DUNE-RB > ./dunerbServer
YaspGridParameterBlock: Parameter 'overlap' not specified, defaulting to '0'.
server: waiting for connections...
server: got connection from 127.0.0.1
```

```
Received call for processing 'init_model'
with 1 arguments and 1 return values.
=====
```

```
read discfunclist_xdr from headerfile, size = 20
Using the explicit ode solver! In order to use a different discretization, change the 'DISCRETIZATION' make variable
```

```
Received call for processing 'get_mu'
with 1 arguments and 1 return values.
=====
```

```
Received call for processing 'rb_symbolic_coefficients'
with 1 arguments and 1 return values.
=====
```

```
█
```

Proof of Concept (linear evolution problem)

```
ans =  
  
lin_evol  
  
>> model.RB_generation_mode  
  
ans =  
  
greedy_uniform_fixed  
  
>> model.RB_stop_Nmax  
  
ans =  
  
20  
  
>> model.T          % this is read from DUNE-RB <<<<  
  
ans =  
  
1  
  
>>  
>>  
>>  
>>  
  >> % generate high dimensional model specific data, like e.g.  
  >> % the grid  
  >>  
  >> model_data = gen_model_data(model);  
  >>
```

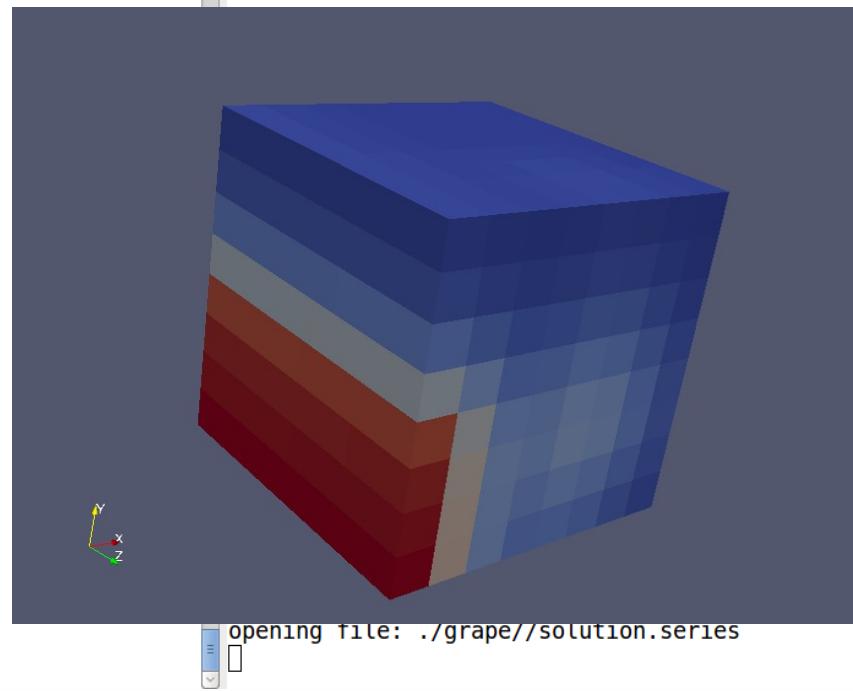
```
DUNE-RB > ./dunerbServer  
YaspGridParameterBlock: Parameter 'overlap' not specified, defaulting to '0'.  
server: waiting for connections...  
server: got connection from 127.0.0.1  
  
Received call for processing 'init_model'  
with 1 arguments and 1 return values.  
=====  
  
read discfunclist_xdr from headerfile, size = 20  
Using the explicit ode solver! In order to use a different discretization,  
change the 'DISCRETIZATION' make variable  
  
Received call for processing 'get_mu'  
with 1 arguments and 1 return values.  
=====  
  
Received call for processing 'rb_symbolic_coefficients'  
with 1 arguments and 1 return values.  
=====  
  
Received call for processing 'gen_model_data'  
with 1 arguments and 1 return values.  
=====
```

Proof of Concept (linear evolution problem)

```
Received call for processing 'rb_symbolic_coefficients'
with 1 arguments and 1 return values.
=====
Received call for processing 'gen_model_data'
with 1 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'detailed_simulation'
with 1 arguments and 1 return values.
=====
opening file: ./grape//solution.series

Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'detailed_simulation'
with 1 arguments and 1 return values.
=====
opening file: ./grape//solution.series
```

Proof of Concept (linear evolution problem)



Proof of Concept (linear evolution problem)

```
>> % Generate the reduced basis with the POD-Greedy algorithm
>> % in DUNE-RB >>>>>
>>
>> detailed_data = gen_detailed_data(model, model_data);
Starting RB extension loop

Detected maximum error prediction 0.044006 for mu=[0.001
    1      0.5]
Extended RB to length 2

Detected maximum error prediction 0.015456 for mu=[0  1
Extended RB to length 3

Detected maximum error prediction 0.012877 for mu=[0
    1      0.5]
Extended RB to length 4

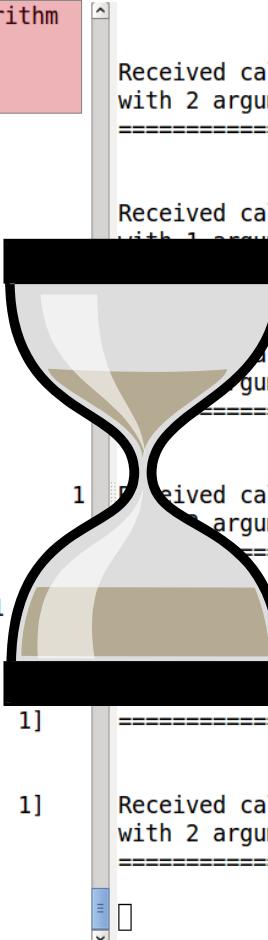
Detected maximum error prediction 0.01064 for mu=[0
    0.5]
Extended RB to length 5

Detected maximum error prediction 0.0084073 for mu=[0.001
    0.5      1]
Extended RB to length 6

Detected maximum error prediction 0.0073233 for mu=[0  1  1]
Extended RB to length 7

Detected maximum error prediction 0.0055443 for mu=[0  1  1]
Extended RB to length 8

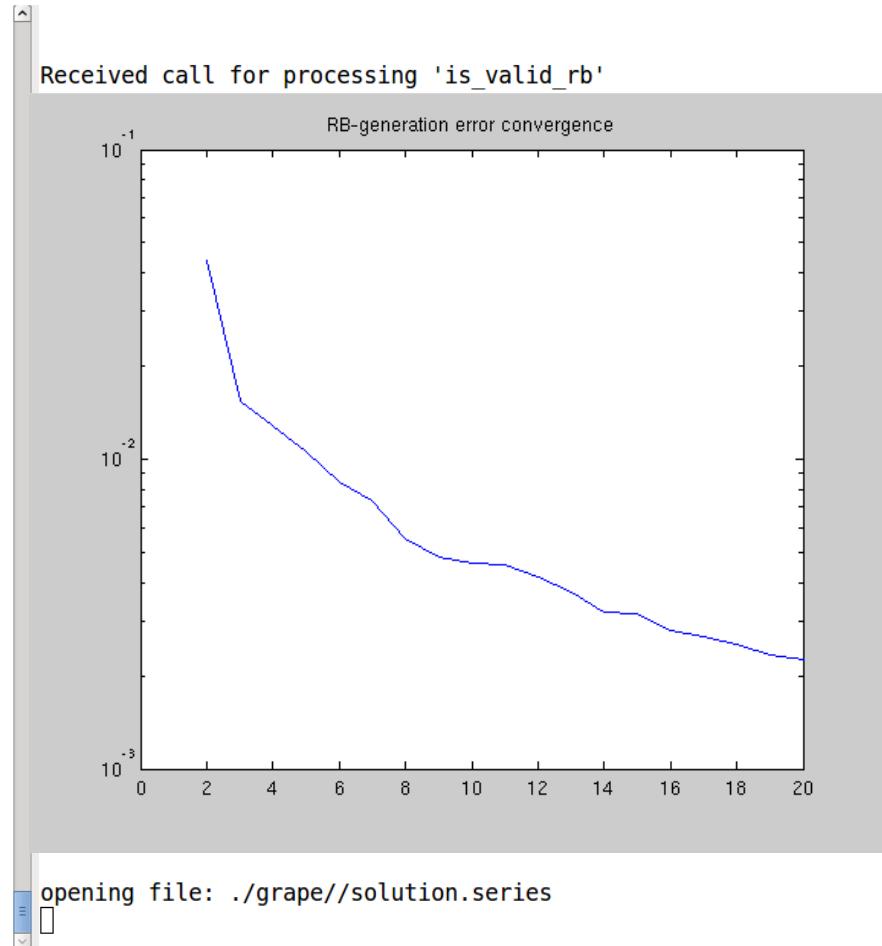
Detected maximum error prediction 0.0048443 for mu=[0
```



```
Received call for processing 'rb_operators'
with 2 arguments and 1 return values.
=====
Received call for processing 'get_mu'
with 1 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 1 arguments and 0 return values.
=====
Received call for processing 'rb_extension_PCA'
with 1 arguments and 1 return values.
=====
: ./grape//solution.series
Received call for processing 'set_mu'
with 1 arguments and 0 return values.
=====
Received call for processing 'rb_init_values'
with 2 arguments and 1 return values.
=====
```

Proof of Concept (linear evolution problem)

```
>>  
>>  
>> detailed_data.RB_info  
  
ans =  
  
    M_train: [3x64 double]  
    max_err_sequence: [1x20 double]  
    mu_sequence: [3x20 double]  
    mu_ind_seq: [1x20 double]  
    toc_value_sequence: [1x20 double]  
    M_first_errs: [64x1 double]  
    stopped_on_epsilon: 0  
stopped_on_max_val_train_ratio: 0  
    stopped_on_timeout: 0  
    stopped_on_Nmax: 1  
    stopped_on_empty_extension: 0  
stopped_on_Nlimit_estimation: 0  
    M_last_errs: [64x1 double]  
elapsed_time: 3.0000e-06
```



Proof of Concept (linear evolution problem)

```
>> % NOW: Generate reduced matrices for online computations
>> % and get them to RBMATLAB <<<<<
>>
>> reduced_data = gen_reduced_data(model, detailed_data)
reduced_data =
    a0: {[1x20 double]}
    LL_I: {2x1 cell}
    LL_E: {5x1 cell}
    bb: {4x1 cell}
    K_II: {4x1 cell}
    K_IE: {10x1 cell}
    K_EE: {25x1 cell}
    m_I: {8x1 cell}
    m_E: {20x1 cell}
    m: {16x1 cell}
    N: 20
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
>>
```

```
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'reconstruct_and_compare'
with 2 arguments and 0 return values.
=====
opening file: ./grape//solution.series
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
```

Proof of Concept (linear evolution problem)

```
>> % The matrices are all of small sizes: e.g. the explicit
>> % discretization operator: LL_E
>>
>> sizes=cellfun(@(X) size(X), reduced_data.LL_E, ...
    'UniformOutput', false);
>> sizes{::}

ans =
20 20

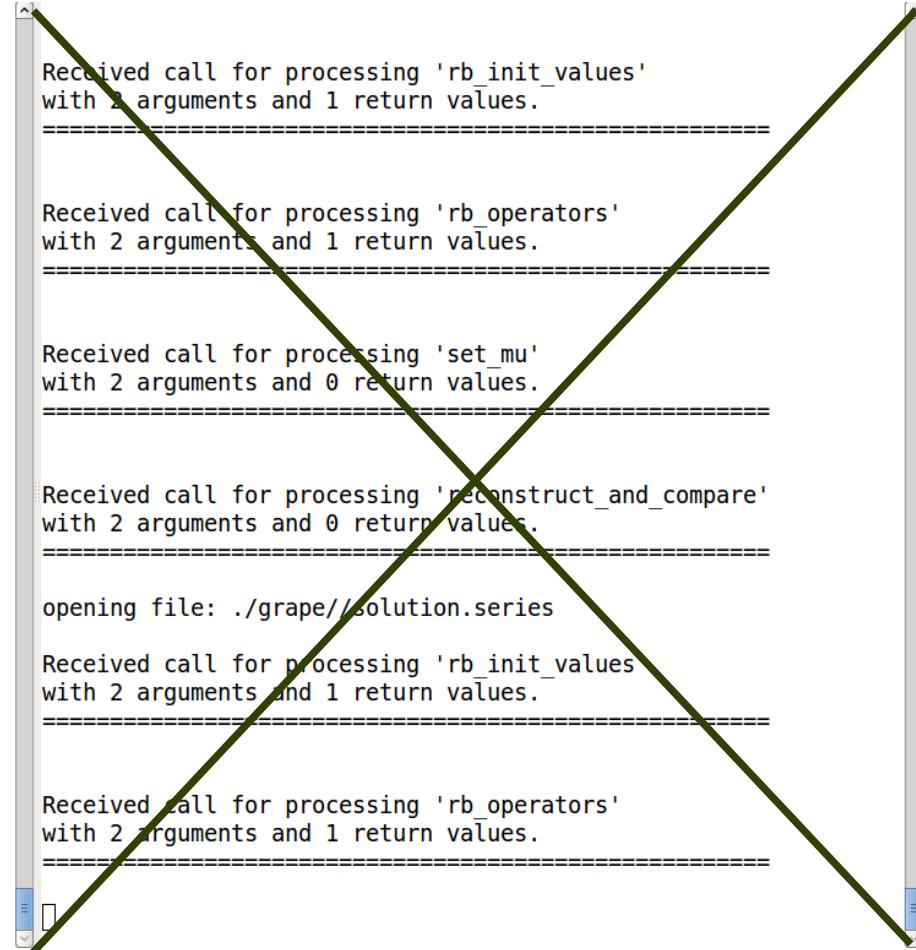
>>
```

```
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'reconstruct_and_compare'
with 2 arguments and 0 return values.
=====
opening file: ./grape//solution.series
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
=====
```

Proof of Concept (linear evolution problem)

```
>> % Now fast reduced simulations are possible in RBMATLAB
>> % without any communication to DUNE-RB
>>
>> model = model.set_mu(model, [0 0.5 1], true);
>>
>> tic; rb_sim_data=rb_simulation(model, reduced_data); toc
Elapsed time is 0.020223 seconds.
>>
>> rb_sim_data
rb_sim_data =
    a: [20x113 double]
    Delta: [1x113 double]
    LL_I: [20x20 double]
    LL_E: [20x20 double]
>>
>>
>> % Error estimator at end time:
>>
>> rb_sim_data.Delta(end)
ans =
    0.0019
>>
>>
>>
>>
```

0.02 seconds!



```
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'reconstruct_and_compare'
with 2 arguments and 0 return values.
=====
opening file: ./grape//solution.series
Received call for processing 'rb init_values'
with 2 arguments and 1 return values.
=====
Received call for processing 'rb operators'
with 2 arguments and 1 return values.
```

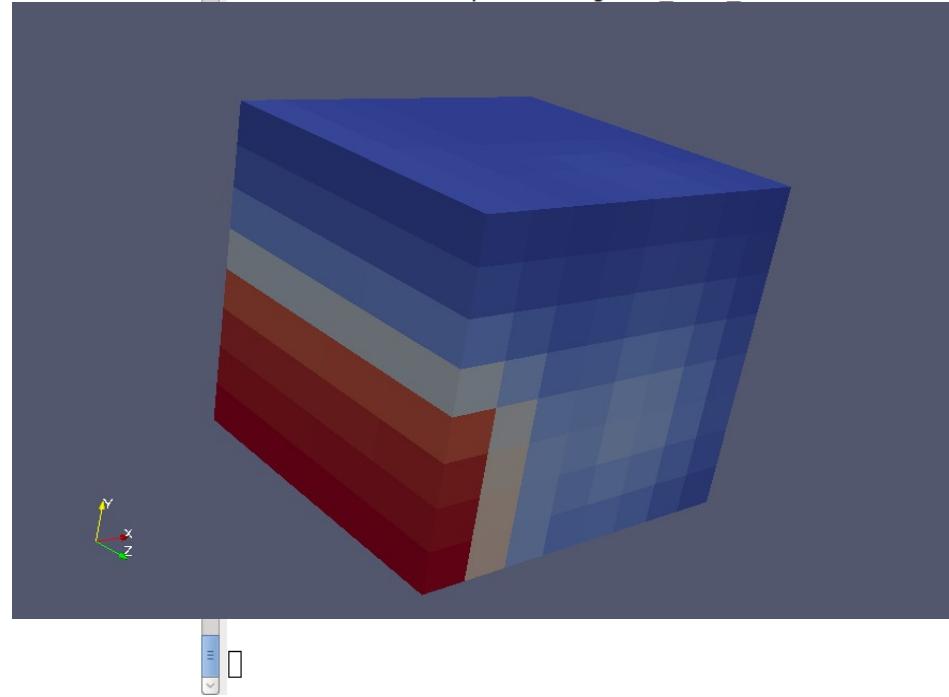
Proof of Concept (linear evolution problem)

```
>> % Of course, the solution can be reconstructed in
>> % DUNE-RB >>>>>>
>>
>> tic;...
dummy=rb_reconstruction(model, detailed_data, rb_sim_data); toc
Elapsed time is 0.359108 seconds.
```

Received call for processing 'reconstruct_and_compare' with 2 arguments and 0 return values.

```
opening file: ./grape//solution.series
```

Received call for processing 'rb init values'



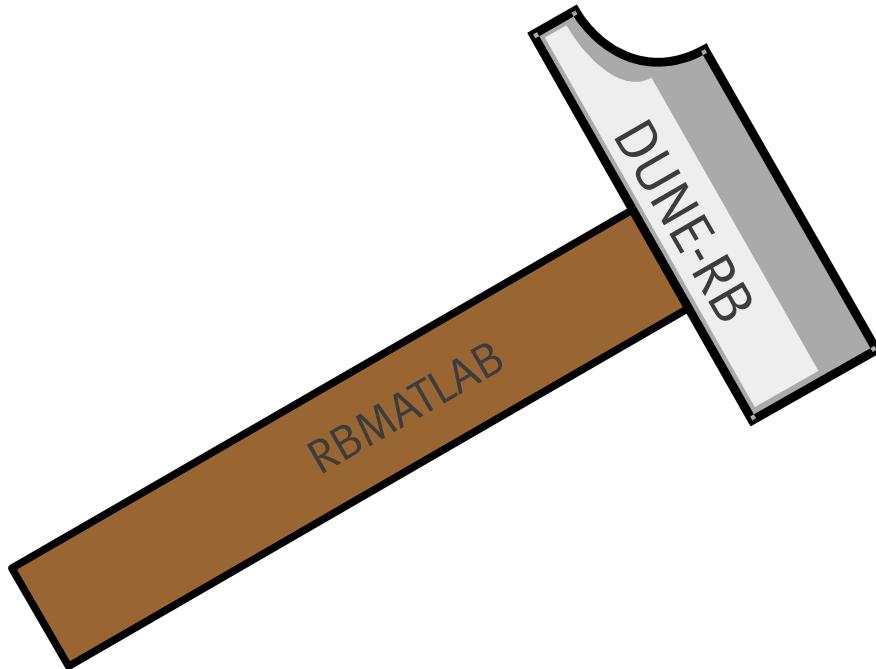
Linear Transport Problem

	High-dim. Solution (sec)	RB Gen- eration	Gen. of Online Matrices	Reduced Sim.	Recon- struction	Grid Cells	$L^\infty - L^2$ - Error ¹
2D Transport (25 Base Functions)	11	71	6.69	0.11	0.33	1,024	$1.42 \cdot 10^{-3}$
2D Transport (50 Base Functions)	11	2,250	21	0.15	0.42	1,024	$4.64 \cdot 10^{-4}$
3D Transport (50 Base Functions)	944	$1.57 \cdot 10^5$	4,659	0.15	26	32,768	$9.11 \cdot 10^{-4}$

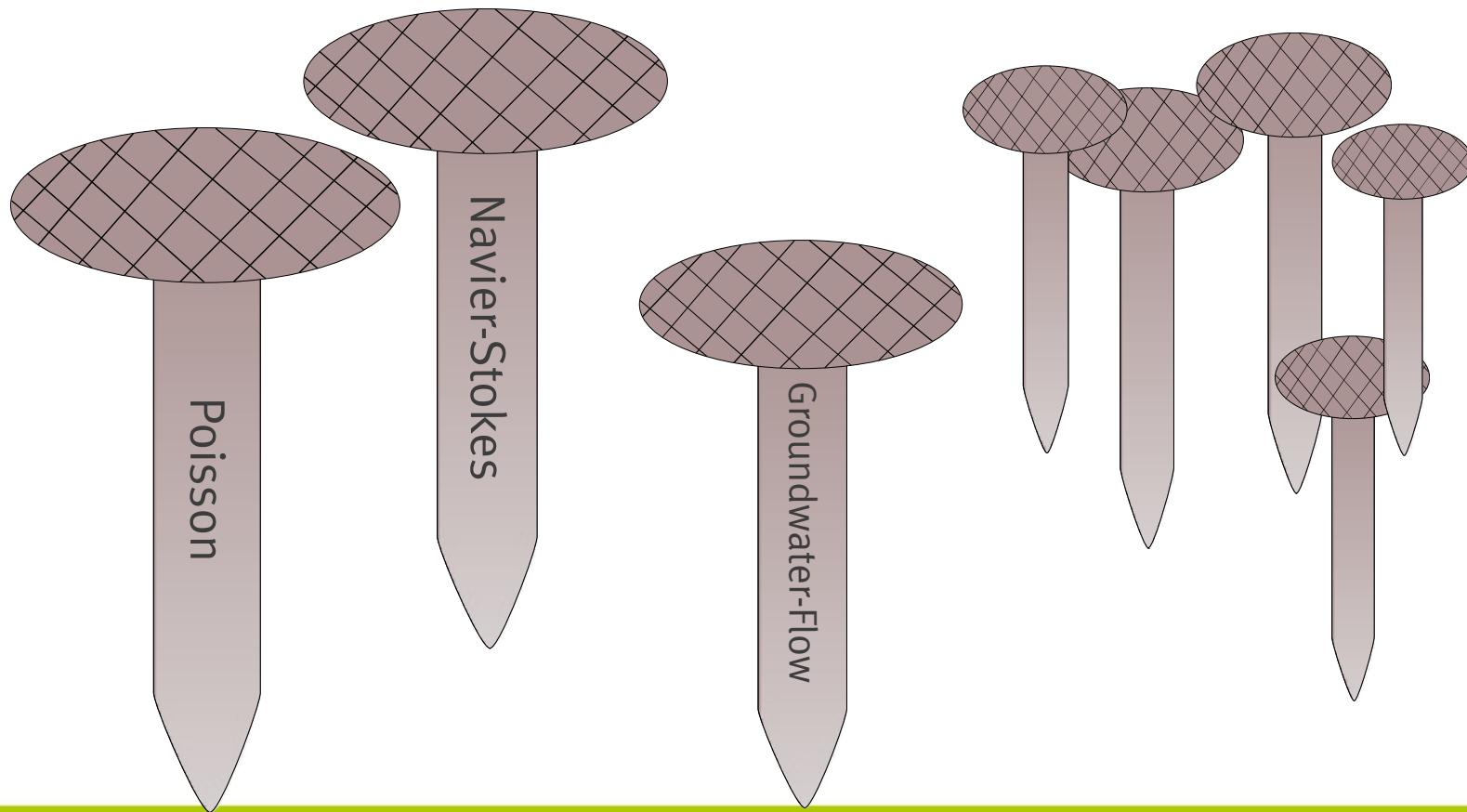
Table 1: Numerical results for a transport problem in 2D and 3D with non-divergence-free velocity.



So, we have a hammer for linear evolution problems!



But how do you make your problems look like a nail?



Interface to PDE descretizations

```
>> % Generate the reduced basis with the POD-Greedy algorithm
>> % in DUNE-RB >>>>>
>>
>> detailed_data = gen_detailed_data(model, model_data);
Starting RB extension loop

Detected maximum error prediction 0.044006 for mu=[0.001
    1      0.5]
Extended RB to length 2

Detected maximum error prediction 0.015456 for mu=[0  1  1]
Extended RB to length 3

Detected maximum error prediction 0.012877 for mu=[0
    1      0.5]
Extended RB to length 4

Detected maximum error prediction 0.01064 for mu=[0
    0.5]
Extended RB to length 5

Detected maximum error prediction 0.0084073 for mu=[0.001
    0.5      1]
Extended RB to length 6

Detected maximum error prediction 0.0073233 for mu=[0  1  1]
Extended RB to length 7

Detected maximum error prediction 0.0055443 for mu=[0  1  1]
Extended RB to length 8

Detected maximum error prediction 0.0048443 for mu=[0
    1      0.5]
```

```
Received call for processing 'rb_operators'
with 2 arguments and 1 return values.
=====
Received call for processing 'get_mu'
with 1 arguments and 1 return values.
=====
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
1: Received call for processing 'rb_extension_PCA'
with 3 arguments and 1 return values.
=====
opening file: ./grape//solution.series
Received call for processing 'set_mu'
with 2 arguments and 0 return values.
=====
Received call for processing 'rb_init_values'
with 2 arguments and 1 return values.
=====
```

Interface to PDE discretizations (linear)

- For PDE discretization of form

For $\mu \in \mathcal{P}$ compute $\{u_h^k(\mu)\}_{k=0}^K \subset \mathcal{W}_h \subset L^2(\Omega)$ by

$$u_h^0(\mu) := P[u_0(\mu)]$$

$$u_h^k(\mu) - \Delta t \mathcal{L}^I(\mu)[u_h^k(\mu)] := u_h^{k-1}(\mu) + \Delta t \mathcal{L}^E(\mu)[u_h^{k-1}(\mu)].$$

- Implement affinely parameter dependent operators

$$\mathcal{L} = \sum_{q=1}^{Q_L} \boxed{\sigma_L^q(\mu)} \boxed{\mathcal{L}^q}$$

Interface to PDE discretizations (linear)

Affinely parametrized operators

$$\mathcal{L} = \sum_{q=1}^{Q_L} \sigma_L^q(\mu) \mathcal{L}^q$$

with methods

- Parametrization [`set_mu()`]
- Parameter dependent **coefficients** [`coefficient()`]
- Parameter independent **operators** [`component()`]

Interface to PDE discretizations (linear)

Affinely parametrized operators

$$\mathcal{L} = \sum_{q=1}^{Q_L} \sigma_L^q(\mu) \mathcal{L}^q$$

with methods

- Parametrization [`set_mu()`]
- Parameter dependent **coefficients** [`coefficient()`]
- Parameter independent **operators** [`component()`]

→ Automatic generation of reduced matrices:

$$(\mathbf{L}^q)_{n,m} = \int_{\Omega} \mathcal{L}^q [\varphi_m] \varphi_n, \quad q = 1, \dots, Q_L, m, n = 1, \dots, N$$

Interface to PDE discretizations (linear)

Affinely parametrized operators

$$\mathcal{L} = \sum_{q=1}^{Q_L} \sigma_L^q(\mu) \mathcal{L}^q$$

with methods

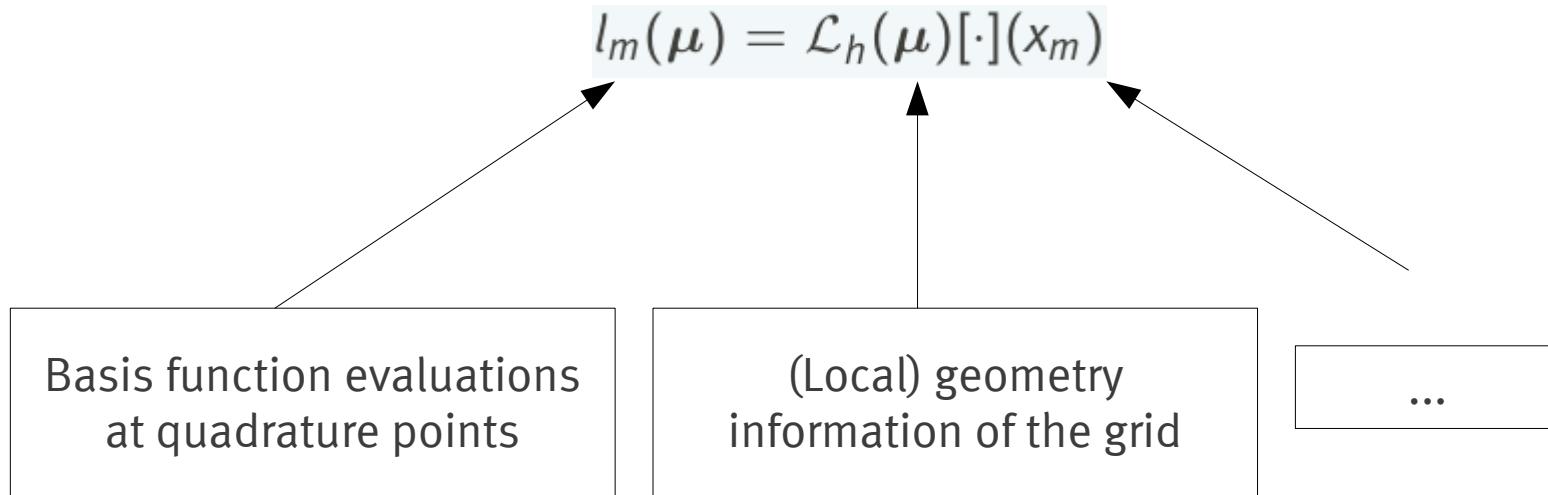
- Parametrization [`set_mu()`]
- Parameter dependent **coefficients** [`coefficient()`]
- Parameter independent **operators** [`component()`]

→ Automatic generation of reduced matrices:

$$(\mathbf{L}^q)_{n,m} = \int_{\Omega} \mathcal{L}^q [\varphi_m] \varphi_n, \quad q = 1, \dots, Q_L, m, n = 1, \dots, N$$

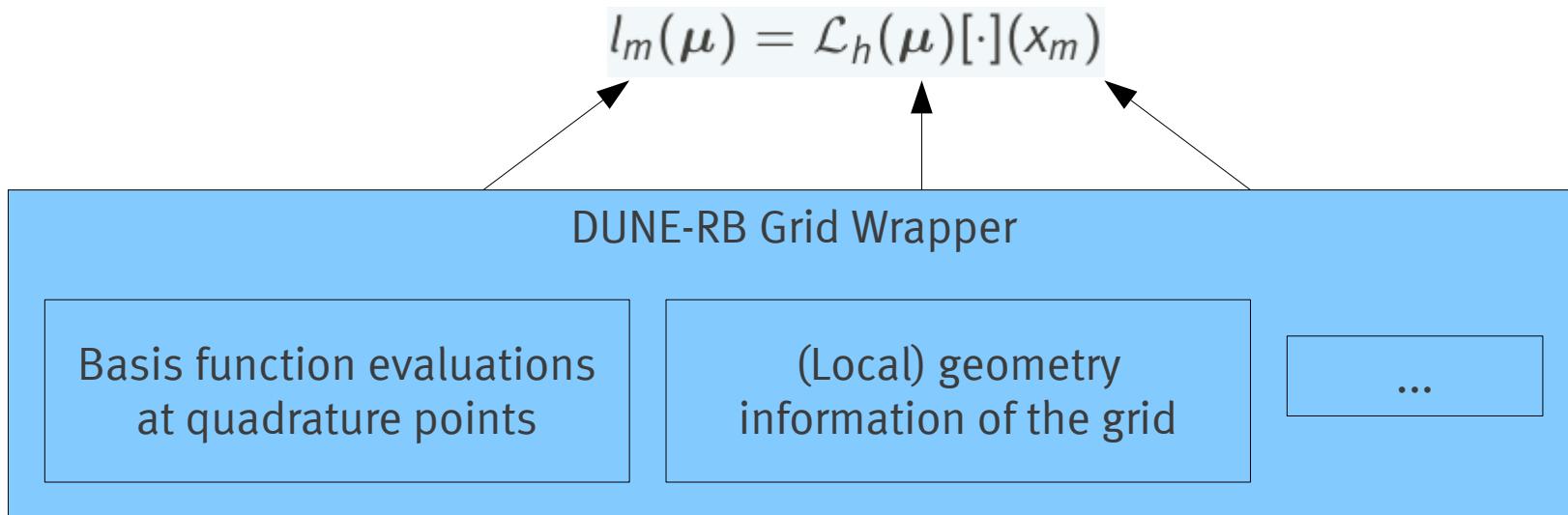
Automatic generation of localized operators

Usual dependencies for local operator evaluations:



Automatic generation of localized operators

Therefore: Delegation of function calls concerning grid geometry and discrete function space to a wrapper:





Behaviour of DUNE-RB Grid Wrapper

- During detailed simulation:

Delegate calls **directly** to the grid and the discrete function space

- During offline phase:

store all grid and discrete function space information concerning the **subgrid** in **low-dimensional** data structures

- During online phase:

Delegate calls to **low-dimensional** data structures